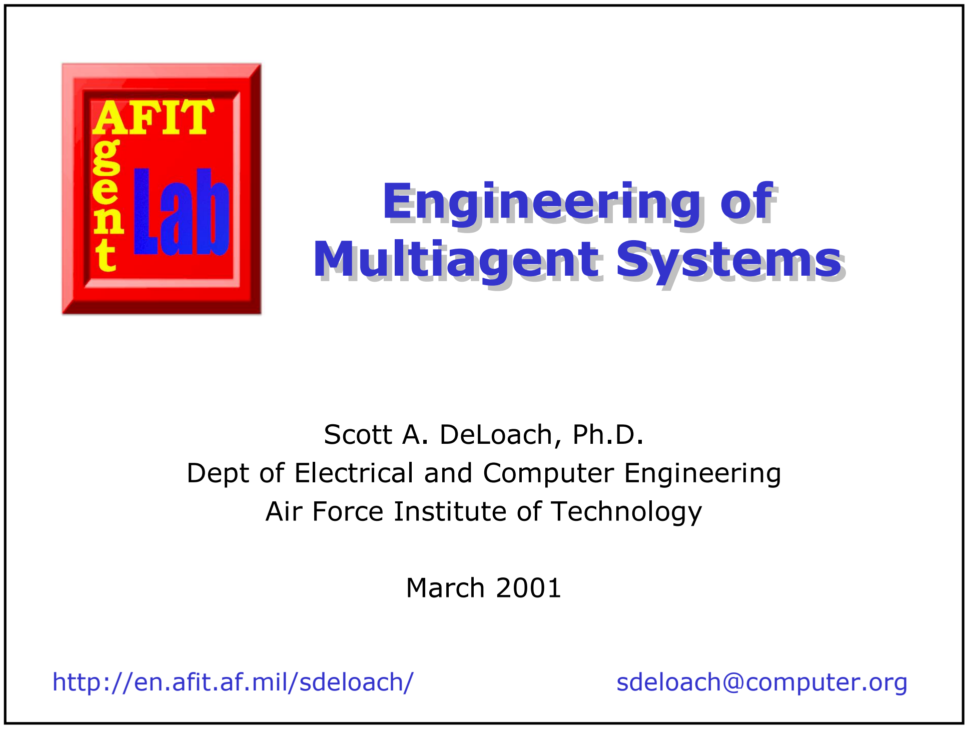
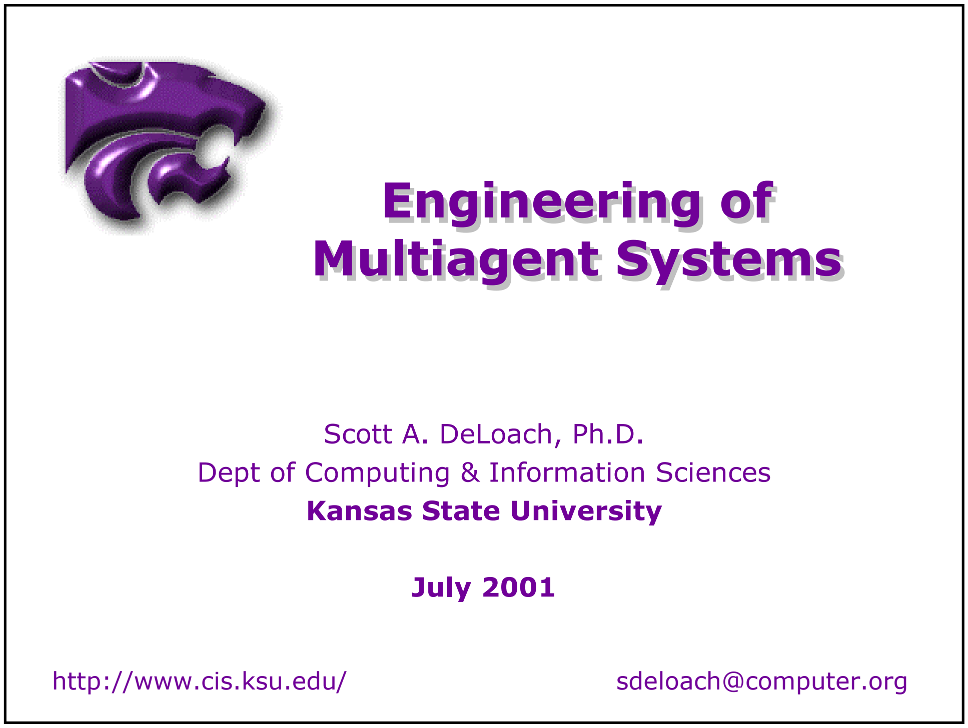


Engineering of Multiagent Systems

Scott A. DeLoach, Ph.D.
Dept of Electrical and Computer Engineering
Air Force Institute of Technology

March 2001

http://en.afit.af.mil/sdeloach/ sdeloach@computer.org

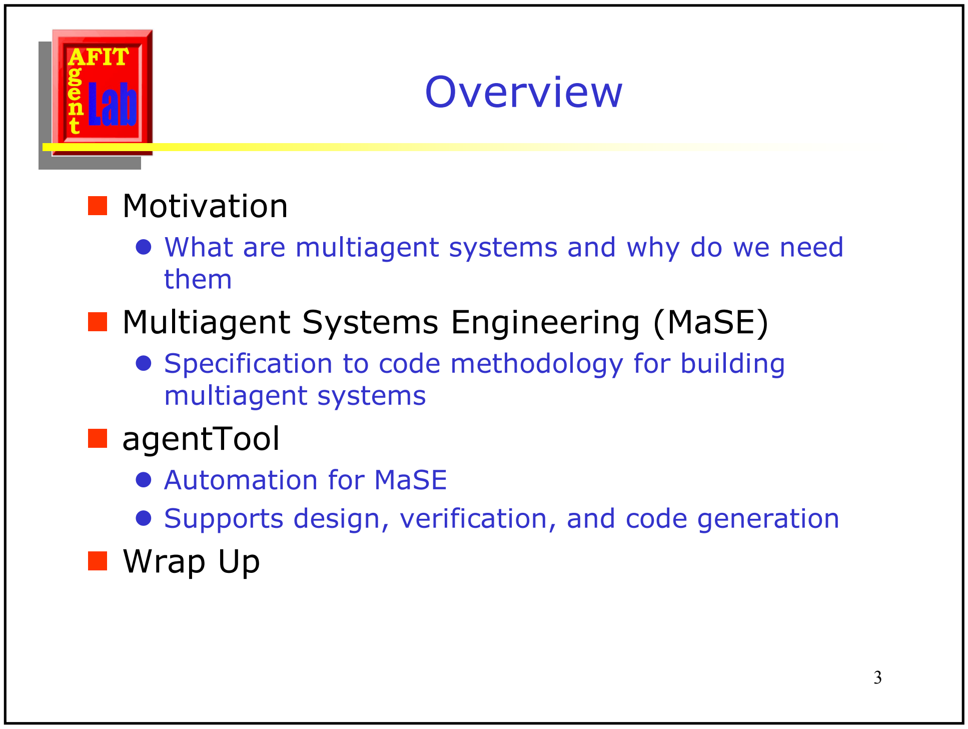


Engineering of Multiagent Systems

Scott A. DeLoach, Ph.D.
Dept of Computing & Information Sciences
Kansas State University

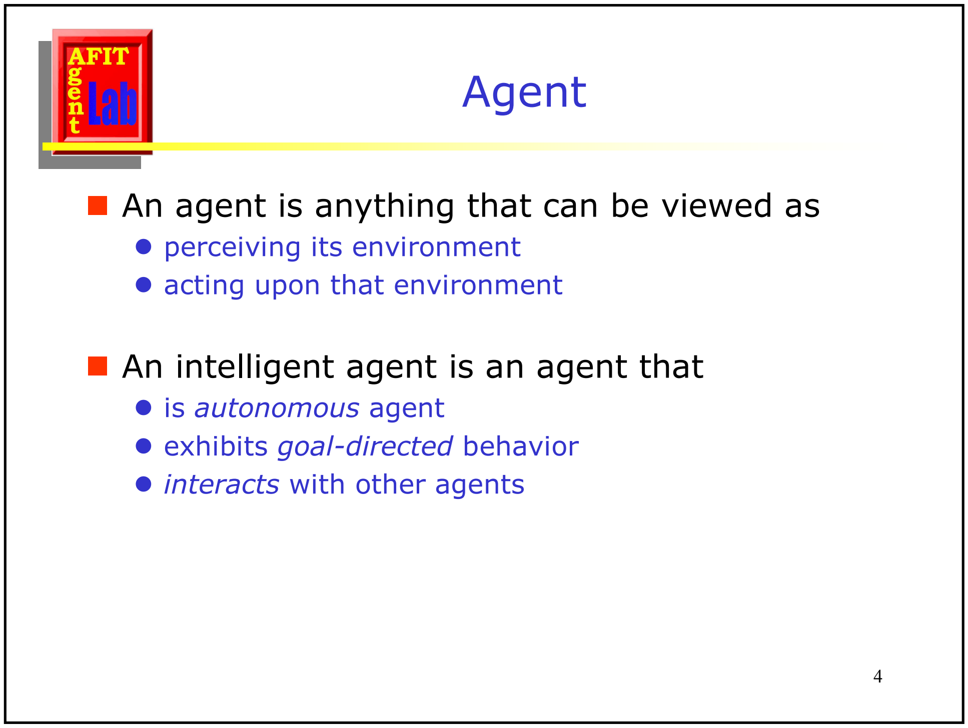
July 2001

http://www.cis.ksu.edu/ sdeloach@computer.org



Overview

- Motivation
  - What are multiagent systems and why do we need them
- Multiagent Systems Engineering (MaSE)
  - Specification to code methodology for building multiagent systems
- agentTool
  - Automation for MaSE
  - Supports design, verification, and code generation
- Wrap Up



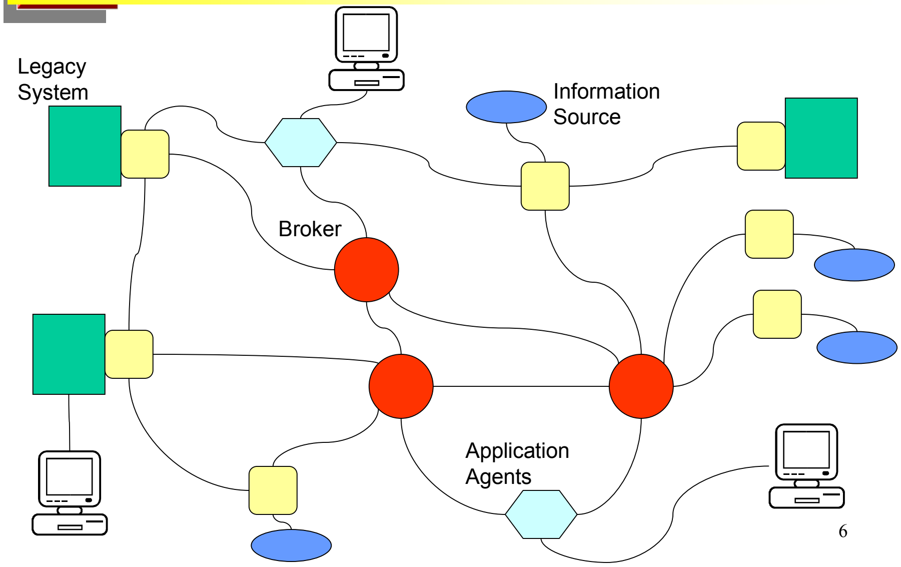
Agent

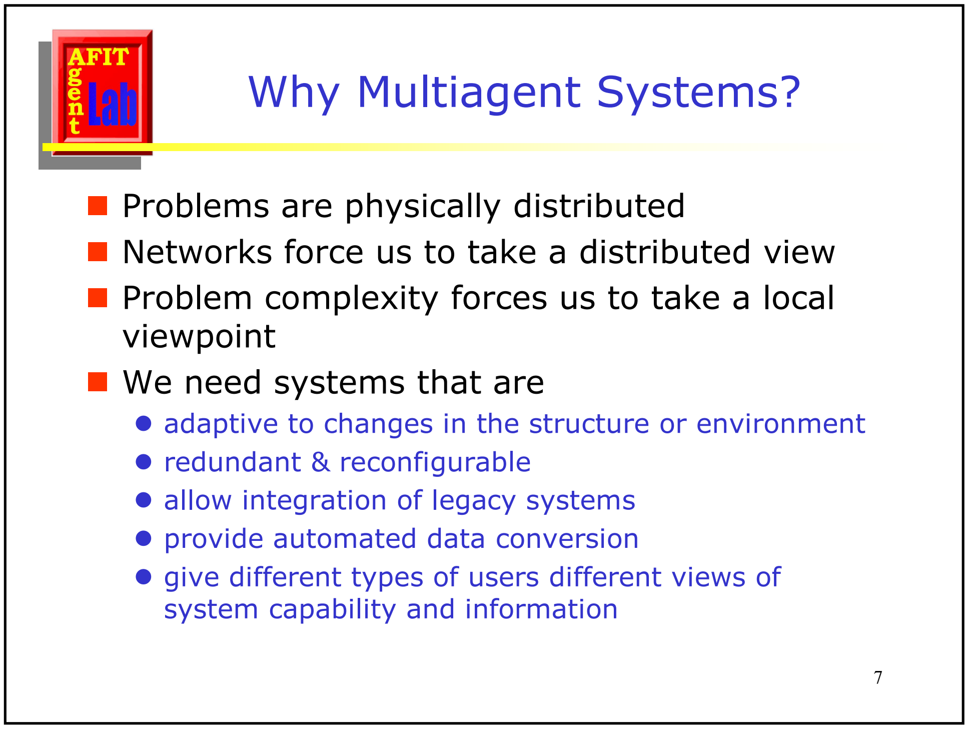
- An agent is anything that can be viewed as
  - perceiving its environment
  - acting upon that environment
- An intelligent agent is an agent that
  - is *autonomous* agent
  - exhibits *goal-directed* behavior
  - *interacts* with other agents

Multiagent System

- A system consisting the following elements
  - An environment
  - A set of objects in the environment
    - Objects are passive and can be perceived, created, destroyed & modified by agents
  - A set of agents
  - A set of relations, which link agents/objects
    - Relations between agents are called acquaintances

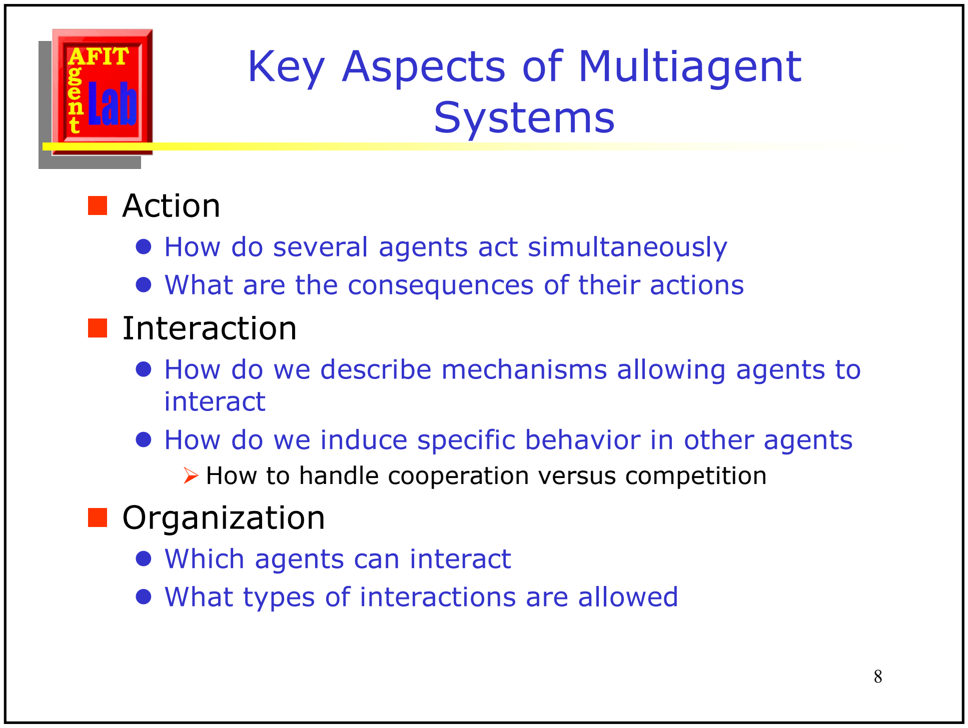
Multiagent Systems





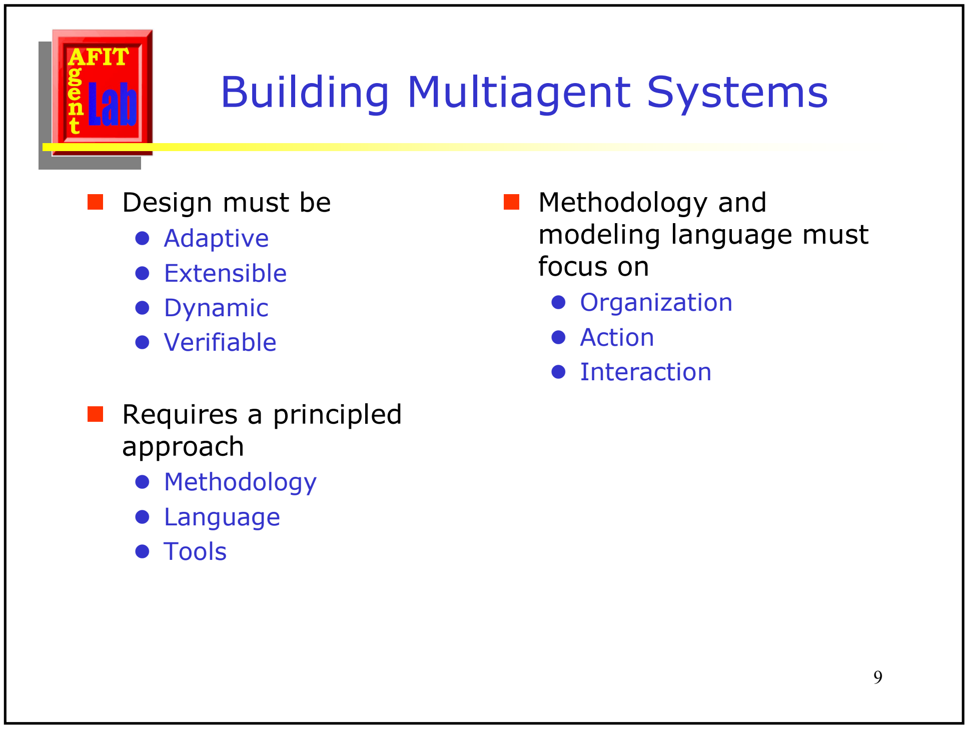
Why Multiagent Systems?

- Problems are physically distributed
- Networks force us to take a distributed view
- Problem complexity forces us to take a local viewpoint
- We need systems that are
  - adaptive to changes in the structure or environment
  - redundant & reconfigurable
  - allow integration of legacy systems
  - provide automated data conversion
  - give different types of users different views of system capability and information



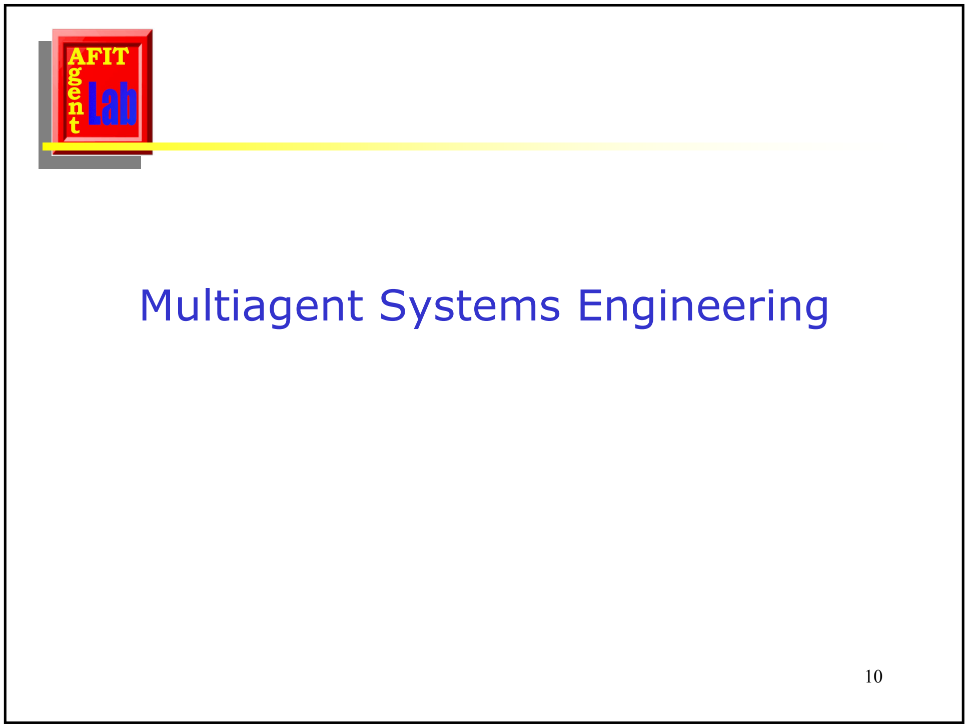
Key Aspects of Multiagent Systems

- Action
  - How do several agents act simultaneously
  - What are the consequences of their actions
- Interaction
  - How do we describe mechanisms allowing agents to interact
  - How do we induce specific behavior in other agents
    - How to handle cooperation versus competition
- Organization
  - Which agents can interact
  - What types of interactions are allowed



Building Multiagent Systems

- Design must be
  - Adaptive
  - Extensible
  - Dynamic
  - Verifiable
- Requires a principled approach
  - Methodology
  - Language
  - Tools
- Methodology and modeling language must focus on
  - Organization
  - Action
  - Interaction



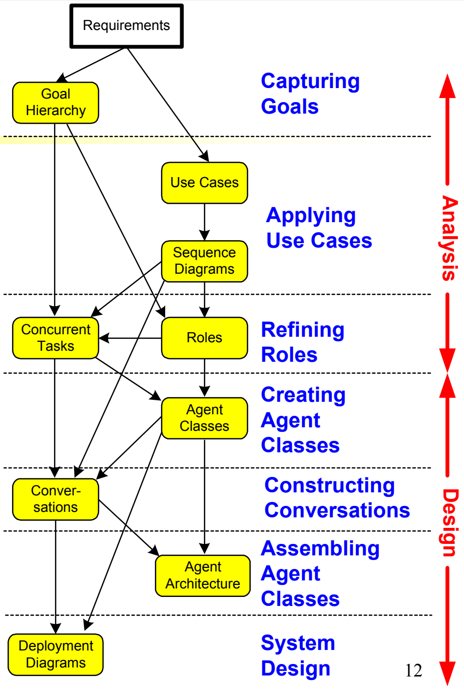
Multiagent Systems Engineering

Methodology Goals

- Full “engineering” approach to multiagent systems development
  - Analysis, design, and implementation
  - Series of graphically based models
  - Logical approach to transforming one model into next
- Support heterogeneous multiagent systems
  - Languages
  - Architectures
  - Environment
- Tool supportable

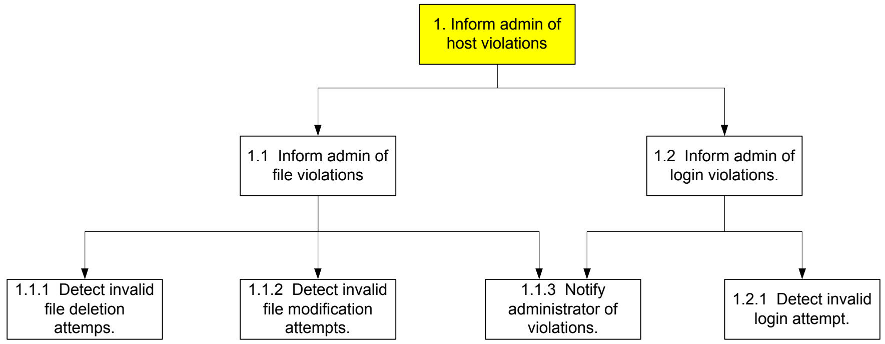
MaSE

- Goal based
- Role based analysis
  - Roles and tasks capture required organization, actions, and interactions
- Roles are played by agent classes
  - Captures organization
- Agent design captures roles and tasks
  - Conversations capture interaction
  - Actions are captured via methods



Capturing Goals

Goal Hierarchy Diagram



1. Inform admin of host violations

1.1 Inform admin of file violations

1.2 Inform admin of login violations.

1.1.1 Detect invalid file deletion attemps.

1.1.2 Detect invalid file modification attempts.

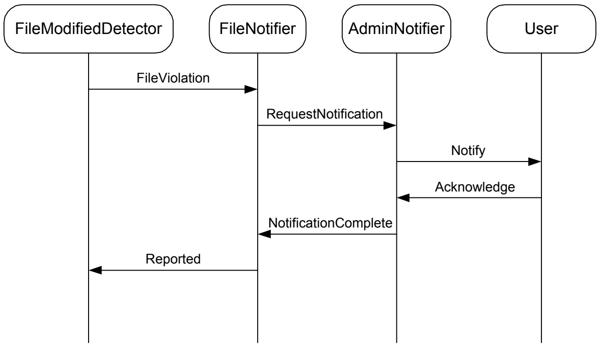
1.1.3 Notify administrator of violations.

1.2.1 Detect invalid login attempt.

Capture the “high-level” goals of the system

Applying Use Cases

Sequence Diagrams



FileModifiedDetector

FileNotifier

AdminNotifier

User

FileViolation

RequestNotification

Notify

Acknowledge

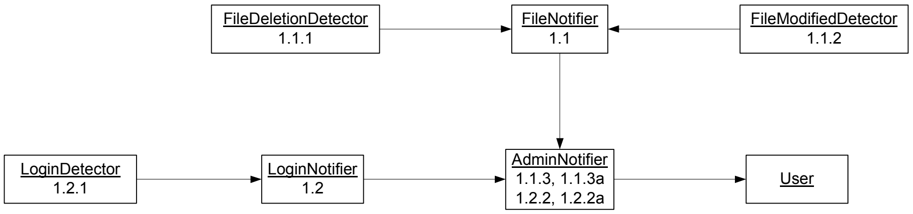
NotificationComplete

Reported

Capture the basic sequence of events between various roles

Refining Roles

Role Model



FileDeletionDetector
1.1.1

FileNotifier
1.1

FileModifiedDetector
1.1.2

LoginDetector
1.2.1

LoginNotifier
1.2

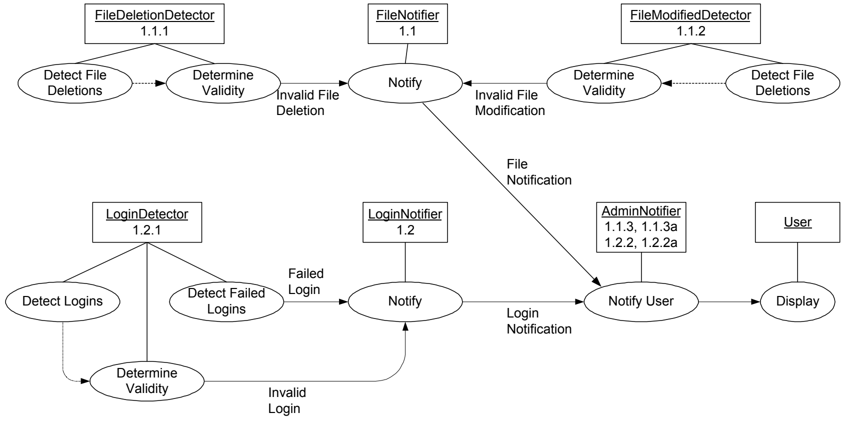
AdminNotifier
1.1.3, 1.1.3a
1.2.2, 1.2.2a

User

Describes all the "roles" that must be played for the system to work as described. Includes notation of interaction relationships

Refining Roles

Extended Role Model



FileDeletionDetector
1.1.1

FileNotifier
1.1

FileModifiedDetector
1.1.2

Detect File Deletions

Determine Validity

Invalid File Deletion

Notify

Invalid File Modification

Determine Validity

Detect File Deletions

File Notification

LoginDetector
1.2.1

LoginNotifier
1.2

AdminNotifier
1.1.3, 1.1.3a
1.2.2, 1.2.2a

User

Detect Logins

Detect Failed Logins

Failed Login

Notify

Login Notification

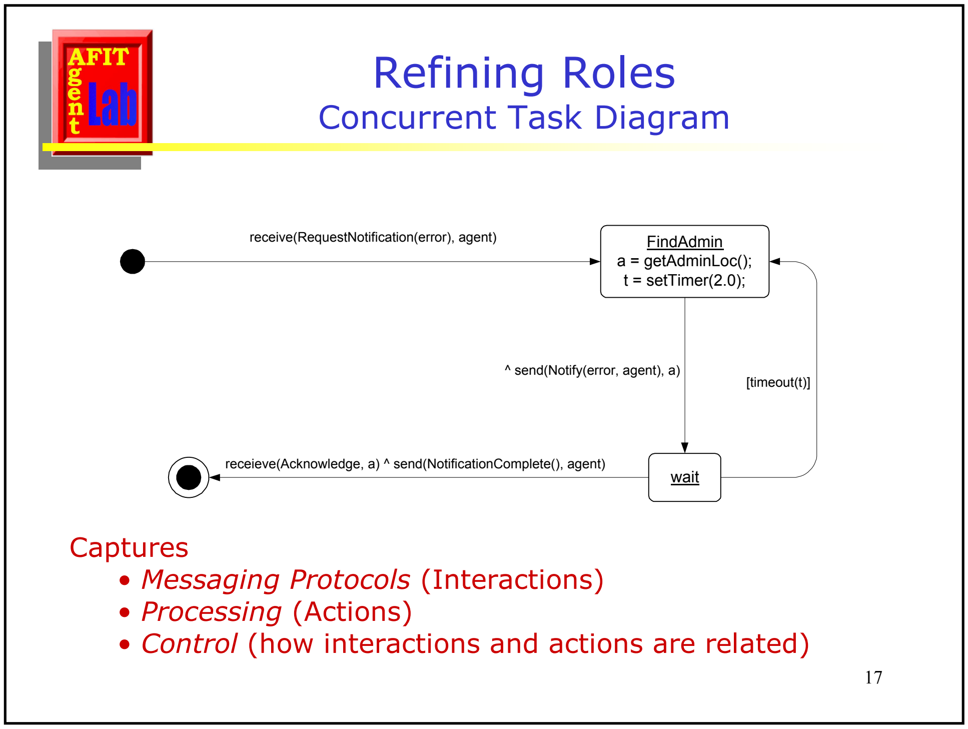
Notify User

Display

Determine Validity

Invalid Login

Adds the concept of *Concurrent Tasks* to capture the behavior required to meet assigned goals and *protocols* to capture the interactions between agents



Refining Roles

Concurrent Task Diagram

receive(RequestNotification(error), agent)

FindAdmin
a = getAdminLoc();
t = setTimer(2.0);

^ send(Notify(error, agent), a)

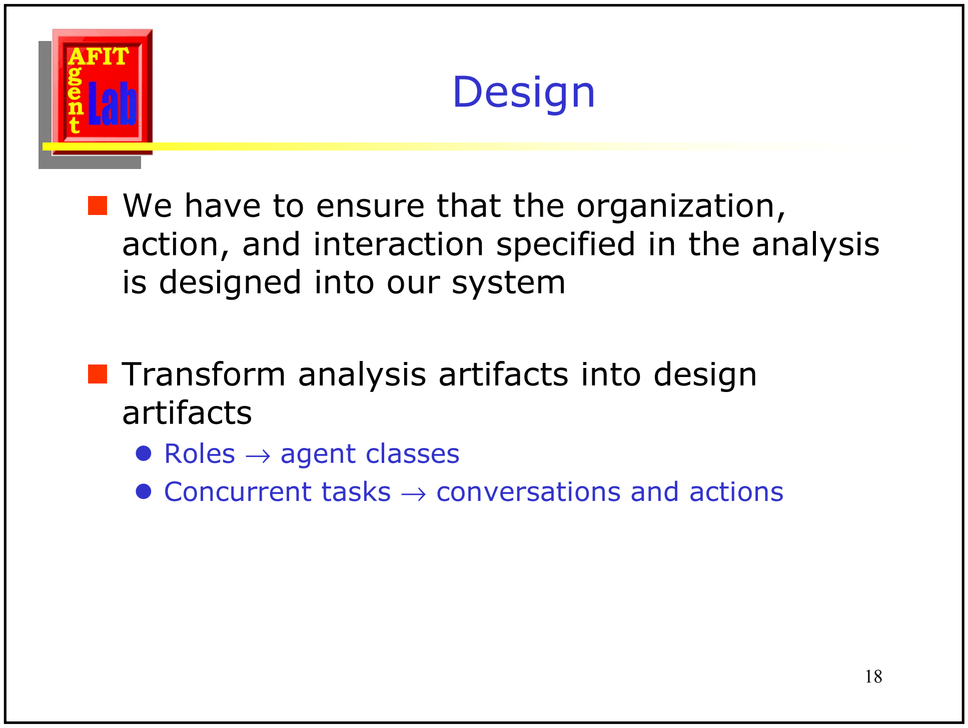
[timeout(t)]

receieve(Acknowledge, a) ^ send(NotificationComplete(), agent)

wait

Captures

- *Messaging Protocols* (Interactions)
- *Processing* (Actions)
- *Control* (how interactions and actions are related)

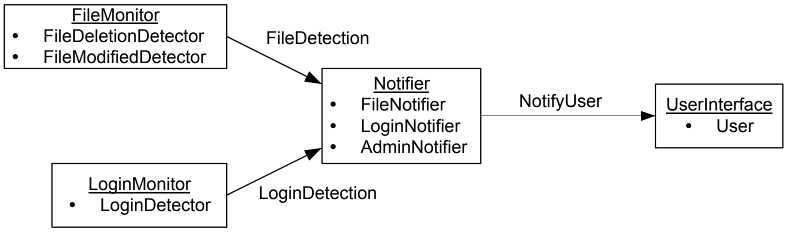


Design

- We have to ensure that the organization, action, and interaction specified in the analysis is designed into our system
- Transform analysis artifacts into design artifacts
  - Roles → agent classes
  - Concurrent tasks → conversations and actions

Creating Agent Classes

Agent Diagram

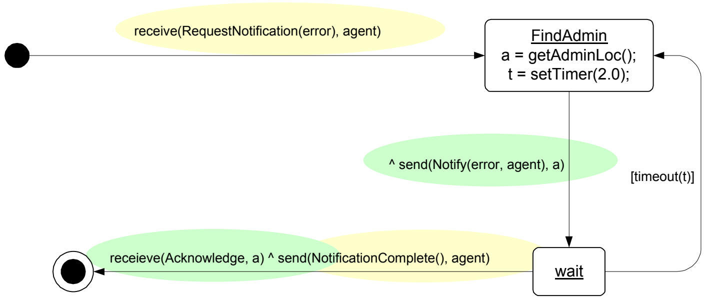


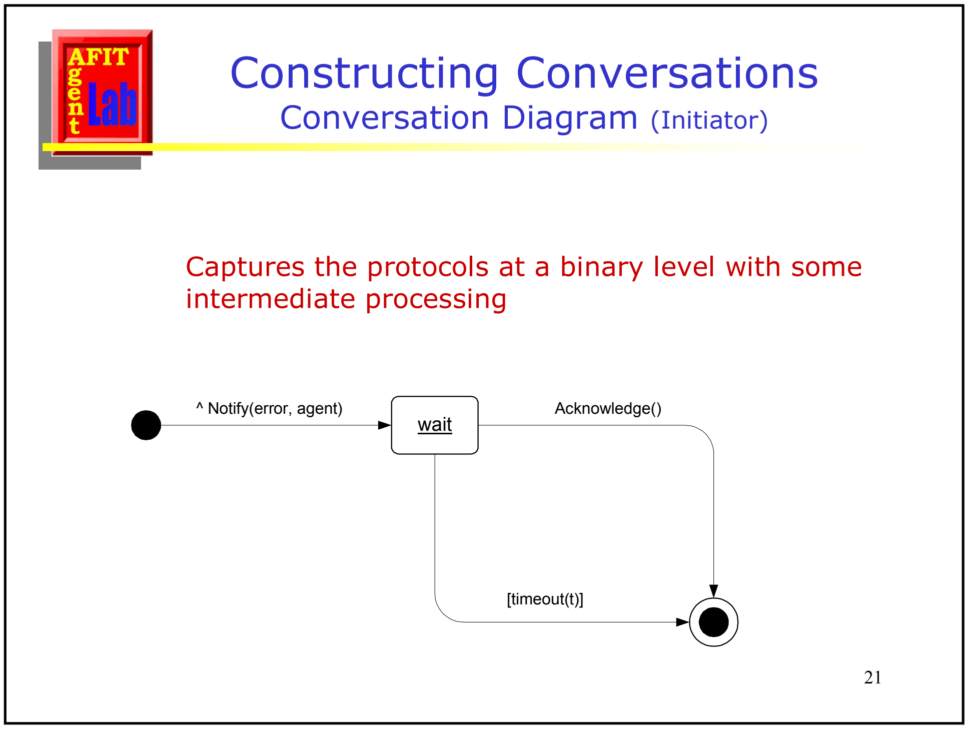
Captures the assignment of roles to *agent classes* and which classes communicate via *conversations.* Shows the overall *organization* of the system

Constructing Conversations

Transforming Concurrent Task Diagrams

Separate protocols into binary conversations between individual agents


receive(RequestNotification(error), agent)
FindAdmin
a = getAdminLoc();
t = setTimer(2.0);
^ send(Notify(error, agent), a)
[timeout(t)]
receieve(Acknowledge, a) ^ send(NotificationComplete(), agent)
wait



Constructing Conversations

Conversation Diagram (Initiator)

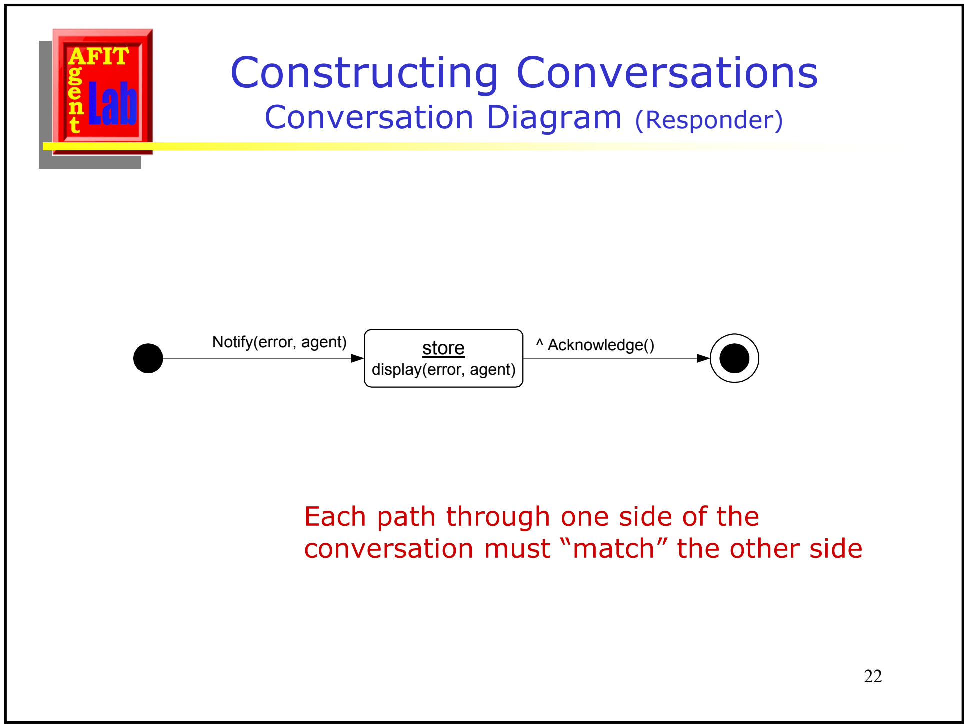
Captures the protocols at a binary level with some intermediate processing

^ Notify(error, agent)

wait

Acknowledge()

[timeout(t)]



Constructing Conversations

Conversation Diagram (Responder)

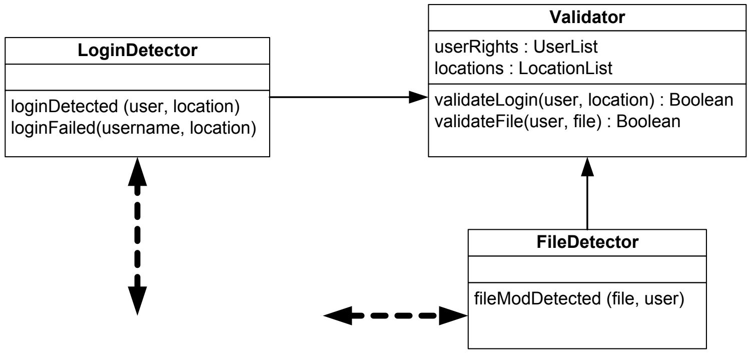
Notify(error, agent)

store
display(error, agent)

^ Acknowledge()

Each path through one side of the conversation must "match" the other side

Assembling Agent Classes



LoginDetector

loginDetected (user, location)
loginFailed(username, location)

Validator

userRights : UserList
locations : LocationList

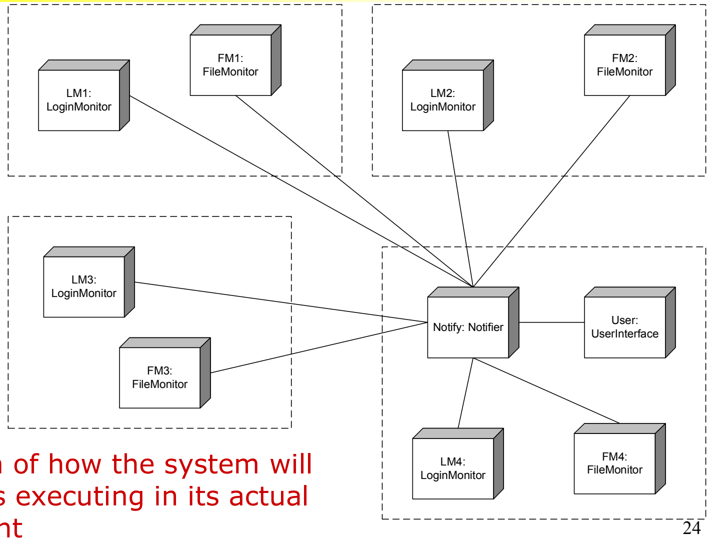
validateLogin(user, location) : Boolean
validateFile(user, file) : Boolean

FileDetector

fileModDetected (file, user)

Defines the internal architecture, methods, and control of individual agents … how an agent carries out *actions*

System Design
Deployment Diagram



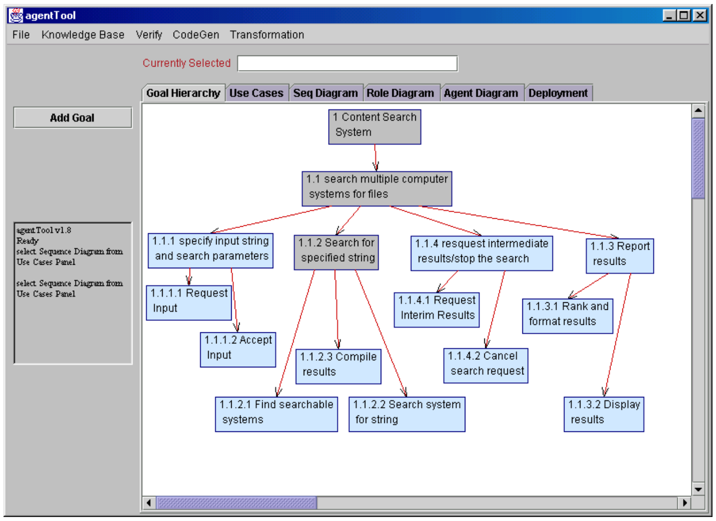
Description of how the system will look as it is executing in its actual environment

agentTool

Toolset Goals

- Enforce methodology
  - Allow users as much freedom as possible
- Automate design transformations
  - Designer performs analysis & makes design decisions
  - Tools carry out details and perform bookkeeping
- Automate verification
  - Verify at the design level before generating code
- Reuse of analysis, design, & code
- Hide formality
- Generate “correct by construction” code

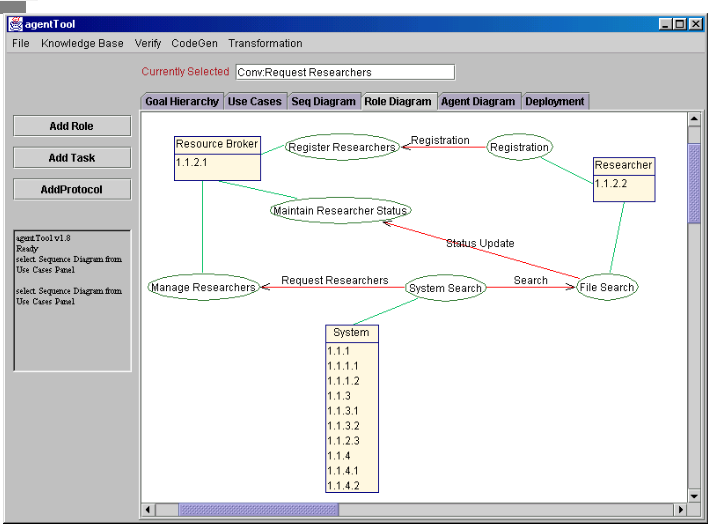
Goal Hierarchy


agentTool
File Knowledge Base Verify CodeGen Transformation
Currently Selected
Goal Hierarchy
Use Cases
Seq Diagram
Role Diagram
Agent Diagram
Deployment
Add Goal
1 Content Search System
1.1 search multiple computer systems for files
1.1.1 specify input string and search parameters
1.1.2 Search for specified string
1.1.4 resquest intermediate results/stop the search
1.1.3 Report results
1.1.1.1 Request Input
1.1.1.2 Accept Input
1.1.4.1 Request Interim Results
1.1.3.1 Rank and format results
1.1.2.3 Compile results
1.1.4.2 Cancel search request
1.1.2.1 Find searchable systems
1.1.2.2 Search system for string
1.1.3.2 Display results

agentTool v1.8
Ready
select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

Role Model

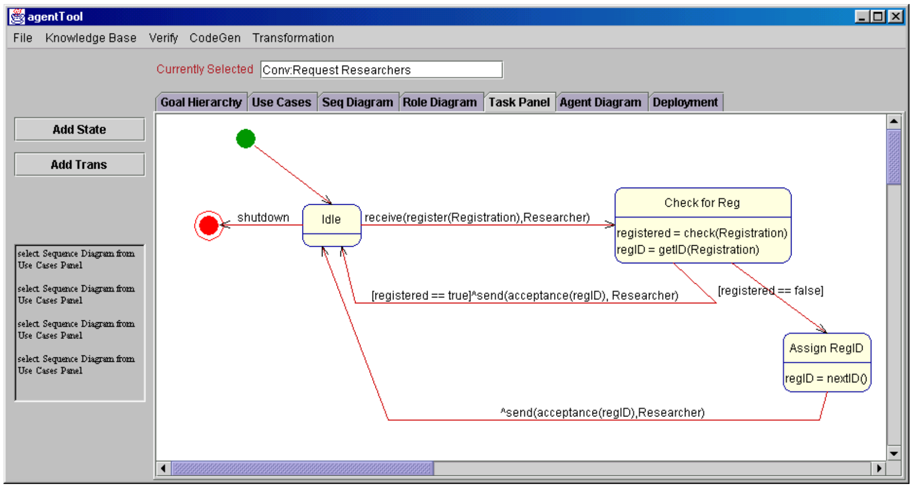


Currently Selected: Conv:Request Researchers

agentTool v1.8
Ready
select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

Concurrent Task Diagram



Currently Selected: Conv:Request Researchers

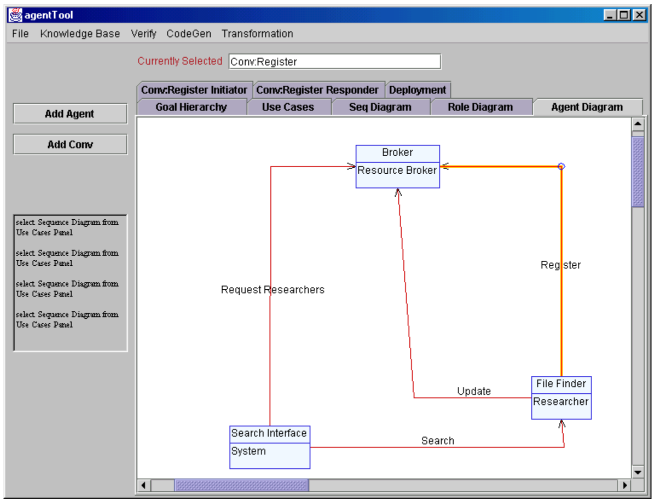
select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

Agent Diagram



Currently Selected: Conv:Register

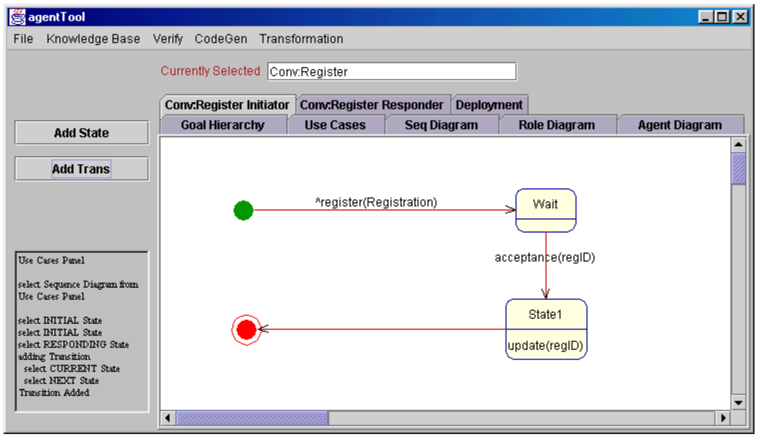
select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

select Sequence Diagram from Use Cases Panel

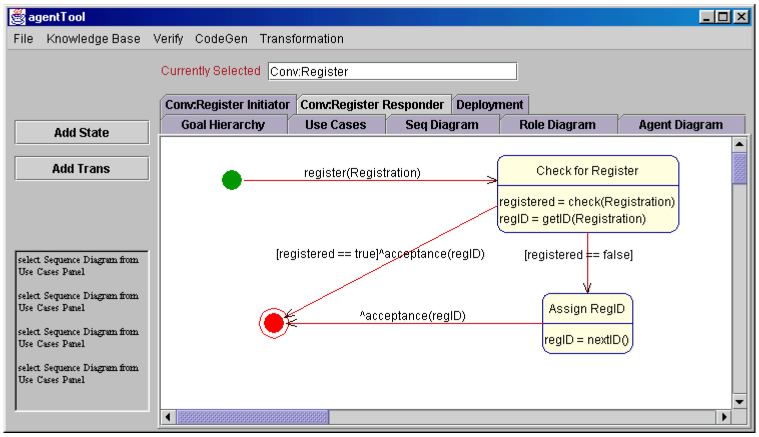
Conversation Diagram



Currently Selected Conv:Register

```
Use Cases Panel

select Sequence Diagram from
Use Cases Panel

select INITIAL State
select INITIAL State
select RESPONDING State
adding Transition
 select CURRENT State
 select NEXT State
Transition Added
```

Conversation Diagram



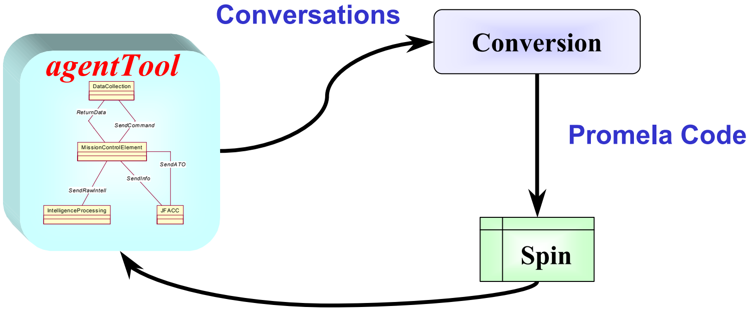
Currently Selected Conv:Register

```
select Sequence Diagram from
Use Cases Panel

select Sequence Diagram from
Use Cases Panel

select Sequence Diagram from
Use Cases Panel

select Sequence Diagram from
Use Cases Panel
```

Support Transformations

- User decides "what" to do
  - What roles should be played by what agent
  - What communication in a task should become a conversation
- agentTool performs automatable tasks
  - Transform tasks communications and activities into conversations and agent methods
- Analysis to design transformations are currently in development

Automate Verification



Conversations

Conversion

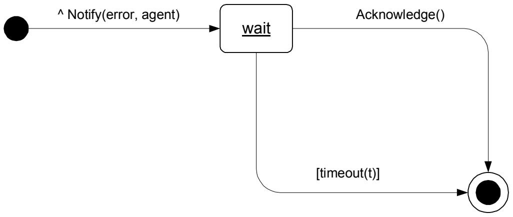
Promela Code

Spin

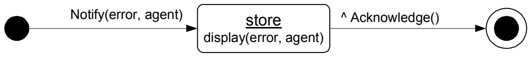
Feedback
- text window
- graphical

Verification Example

Initiator

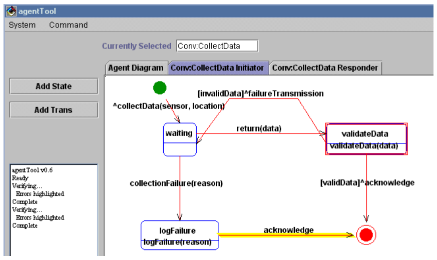

^ Notify(error, agent)
wait
Acknowledge()
[timeout(t)]

Responder


Notify(error, agent)
store
display(error, agent)
^ Acknowledge()

Feedback

```
proc 0 = :init:
proc 1 = SendInfoInitiator
proc 2 = SendInfoResponder
proc 3 = CollectDataInitiator
proc 4 = CollectDataResponder
q\p   0   1   2   3   4
  1   .   .   .   CollectData!collectData
  1   .   .   .   .   CollectData?collectData
  1   .   .   .   .   CollectData!collectionFailure
  1   .   .   .   CollectData?collectionFailure
  2   .   SendInfo!send
  2   .   .   SendInfo?send
  2   .   .   SendInfo!acknowledge
  2   .   SendInfo?acknowledge
spin: trail ends after 16 steps
-------------
final state:
-------------
#processes: 5
 16:           proc  4 (CollectDataResponder) line  92 "verify" (state 27)
proc  3 (CollectDataInitiator) line  65 "verify" (state 24)
proc  2 (SendInfoResponder) line  46 "verify" (state 24) <valid endstate>
proc  1 (SendInfoInitiator) line  25 "verify" (state 22) <valid endstate>
proc  0 (:init:) line 114 "verify" (state 6) <valid endstate>
5 processes created
```

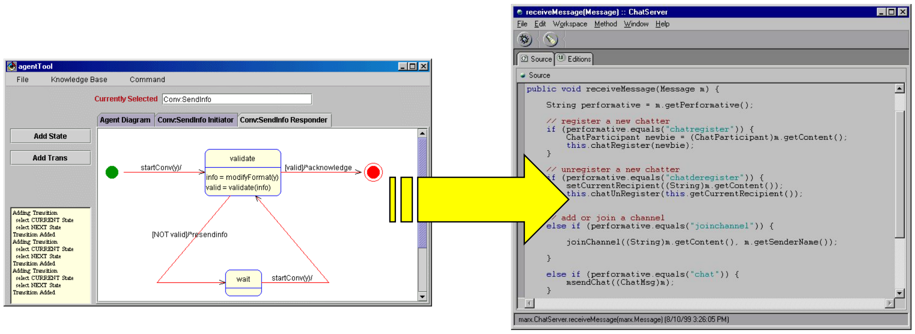


```
agentTool v0.6
Ready
Verifying...
  Errors highlighted
Complete
Verifying...
  Errors highlighted
Complete
```

```
DEADLOCK CONDITION EXISTS IN THE FOLLOWING CONVERSATION:
Conversation Name = CollectData
Participant Name = Responder
Current State = wait
State Transition = acknowledge

DEADLOCK CONDITION EXISTS IN THE FOLLOWING CONVERSATION:
Conversation Name = CollectData
Participant Name = Initiator
Current State = logFailure
State Transition = acknowledge
```

Code Generation

- Automatic from agent and conversation diagrams
- Select platform-dependent components such as a messaging framework



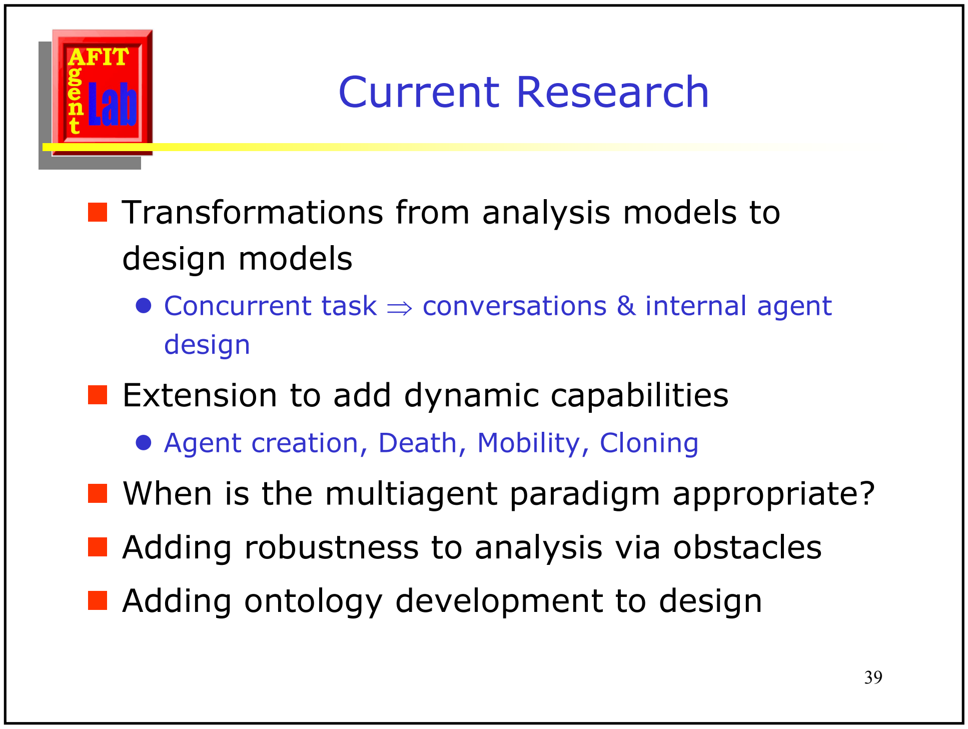
```
Adding Transition
  select CURRENT State
  select NEXT State
Transition Added
Adding Transition
  select CURRENT State
  select NEXT State
Transition Added
Adding Transition
  select CURRENT State
  select NEXT State
Transition Added
```

```
public void receiveMessage(Message m) {

    String performative = m.getPerformative();

    // register a new chatter
    if (performative.equals("chatregister")) {
        ChatParticipant newbie = (ChatParticipant)m.getContent();
        this.chatRegister(newbie);
    }

    // unregister a new chatter
    if (performative.equals("chatderegister")) {
        setCurrentRecipient((String)m.getContent());
        this.chatUnRegister(this.getCurrentRecipient());

    // add or join a channel
    else if (performative.equals("joinchannel")) {

        joinChannel((String)m.getContent(), m.getSenderName());

    }

    else if (performative.equals("chat")) {
        msendChat((ChatMsg)m);
    }
```

marx.ChatServer.receiveMessage(marx.Message) (8/10/99 3:26:05 PM)

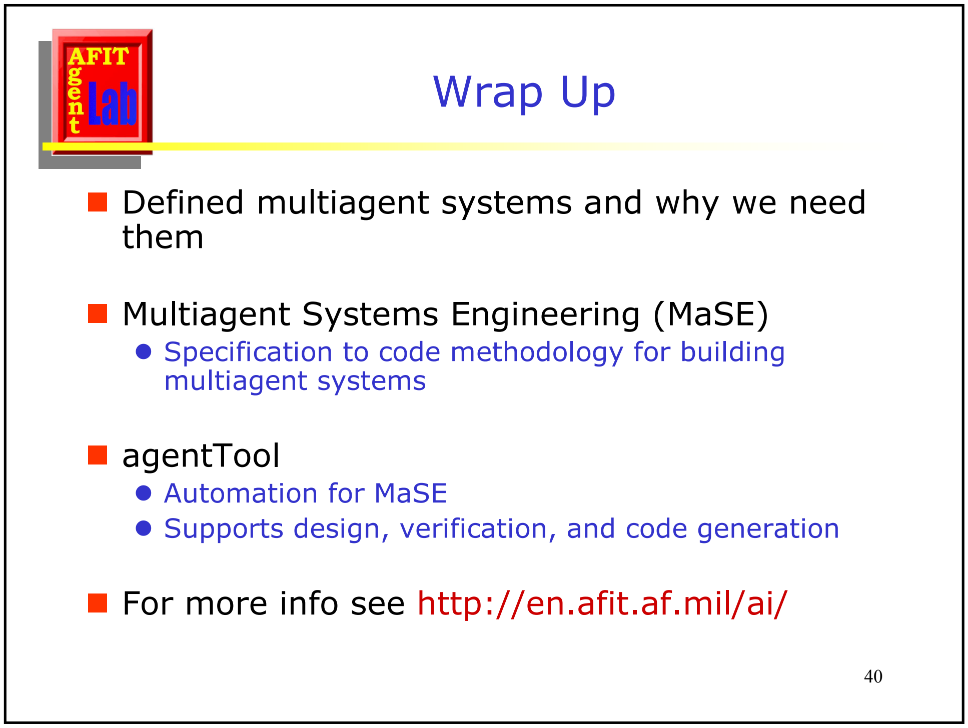
Results

- MaSE and agentTool have been used to develop several small to medium sized multiagent systems
  - Information systems
  - Mixed-initiative distributed planners
  - Biologically-based immune & intrusion detection systems
  - Autonomous control of Uninhabited Air Vehicles
- Users report that MaSE is relatively simple, yet flexible enough to allow a variety of solutions
- Currently developing larger scale multiagent systems that are both mobile and dynamic in nature
- agentTool has an active user list of over 100 world-wide academic, government, and industry users



Current Research

- Transformations from analysis models to design models
  - Concurrent task ⇒ conversations & internal agent design
- Extension to add dynamic capabilities
  - Agent creation, Death, Mobility, Cloning
- When is the multiagent paradigm appropriate?
- Adding robustness to analysis via obstacles
- Adding ontology development to design



Wrap Up

- Defined multiagent systems and why we need them
- Multiagent Systems Engineering (MaSE)
  - Specification to code methodology for building multiagent systems
- agentTool
  - Automation for MaSE
  - Supports design, verification, and code generation
- For more info see http://en.afit.af.mil/ai/